Introduction to OpenBACH-extra and OpenBACH API

Overview

OpenBACH-extra is a tool-box for OpenBACH that contains a set of additional tools that are described in this page.

```
openbach-extra/
├── external_jobs/
├── executors/ -> scripts ready to be launched from CLI: allow to run scenarios or create
scenario JSONs (to be imported in the OpenBACH web interface).
|   ├── reference/ -> executors maintained by the OpenBACH Team
|   └── examples/  -> other examples of executors
└── apis/
    ├── data_access/
    ├── auditorium_scripts/
    └── scenario_builder/
        ├── helpers/ -> predefined blocks of openbach functions that can be imported in
scenarios.
        └── scenarios/ -> reference scenarios to be imported in your scenarios.
```

OpenBACH-extra is stored on the forge.

Cloning OpenBACH-extra on a machine can help manage multiple OpenBACH platforms.

External jobs

Some jobs are installed by default on an OpenBACH platform.

The external jobs of the OpenBACH-extra project are a set of jobs that must be added to an OpenBACH controller for being used on a specific OpenBACH plateform.

This can be done through:

- a Web Interface (see Manage Jobs in the Web Interface Administrator Manual)
- a Command Line Interface (see the Command Line Interface Administrator Manual)

Executors

The executors allow you to straightforward launch scenarios from CLI or import the JSON (thanks to auditorium-scripts API). Their name should begin with *executor_* (for example *executor_network_delay.py*).

A simple usage of the executors is described in the Simple Command Line Interface User Manual.

OpenBACH API

Three tools are available in the API:

- Scenario builder: is a Python tool allowing to build scenarios programmatically and generate JSON files that can be imported into a project of an existing OpenBACH installation.
- Auditorium-scripts: a group of Python scripts equivalent to functions described on the web interface manual. This scripts can be launched from the CLI or imported to your own scripts.
- Data Access: is a Python tool used to access data associated to the various scenario instances of an OpenBACH platform. Optionally, the data access can be used to modify, create, or delete such data.

Setting up your environment for using executors and API

Install the API

The tools and scripts that can be found in the folders of the executors and the API can be used to pilot different OpenBACH platforms.

To use them, the OpenBACH API should be installed as follows on a machine that has access to the OpenBACH platforms it wishes to pilot.

The three tools of the OpenBACH API (scenario builder, auditorium-scripts and data-access) are part of the OpenBACH-API repository and are installed by default on the agents of any OpenBACH platform. You can also install them manually on other machines by cloning the openbach-extra repository and using the `setup.py` file:

```
$ sudo apt-get install python3-setuptools python3-dev build-essential
$ git clone https://forge.net4sat.org/openbach/openbach-extra.git
$ cd openbach-extra/apis/
$ sudo python3 setup.py install
```

Or using `pip` and asking it to install the openbach-api following wheel.

```
$ pip3 install
https://forge.net4sat.org/openbach/openbach/raw/master/pip_mirror/python3/openbach_api-3.x.x-py
3-none-any.whl
```

Use your local (modified) version of the API

When developing new scenarios and helpers, you may want to install your latest version of the API to conduct your tests. Using `python3 setup.py install` to deploy your version as suggested previously may or may not work depending on your configuration. Here are some tips to get you going:

- Due to the way Python's import system works, any package installed by `pip` will take priority over the same package installed by `setup.py`; so if you're testing on an agent, you may want

to `pip uninstall openbach-api` before running `python3 setup.py install`;
- Alternatively, you can force Python to look into your development folder before anything else by using the PYTHONPATH environment variable (*e.g.* `PYTHONPATH=~/openbach-extra/apis/ python3 executor_network_delay.py –arguments… run`);
- Or you can use virtual environments to completely separate your system from your test environment; once the virtual environment is activated, you can safely `python setup.py install` within it to always access your last version when testing (and deactivate it when done).

Once your scenarios/helpers are finished and ready to be committed, you should then modify the version argument (e.g. version='3.1.4') in your setup.py, and commit/push the new scenarios/helpers as well as the setup.py.

This procedure is also recommended when you want to be sure that you are using the last version of the scenarios/helpers.

Description of the controller

Each command must be run using Python 3 and understand the `-h` flag to display the expected arguments. On top of the positional and optional arguments that are unique for each command, all have the following arguments needed by the controller:

- `controller`: the address of the controller to run the command on;
- `login` or `username`: the name to use to connect as on the controller. If this option is provided, the associated password will be asked on the command line when running the command. By default, the username and the password are “openbach”.

Instead of arguments, you can pass these information by means of a file. For that, it is necessary to create a JSON file named “controller” (in the folder you will be running scripts from) with the controller ip, your login and your password, as follows:

```
{
    "controller": "192.168.1.3",
    "login": "openbach",
    "password": "openbach"
}
```

In case you don't specify a password, neither on the command-line nor in the `controller` file, an echo-less prompt will ask it once you run the script.

A note on PYTHONPATH and the usage of multiple OpenBACH platforms

To ease the use of the auditorium scripts with different OpenBACH platforms, it can be better to keep a separate “scripting” directory that will host the “controller” file as well. To use the API scripts, the PATH variable can be changed (in .bashrc for instance or in a particular script):

```
$ export PATH=$PATH:~/openbach-extra/apis/
```

This way, the commands can be used in any directories. The controller file is searched in the local directory where your program is launched.

Do not forget the PYTHONPATH variable either in case you didn't install the API system-wide using `setup.py`. To be able to import the `auditorium_scripts` module from the various scripts, you can:

```
$ export PYTHONPATH=$PYTHONPATH:~/openbach-extra/apis/
```

A note on proxies

The auditorium scripts make use of a tool that understand the HTTP_PROXY and HTTPS_PROXY environment variables. If your platform is not accessible through your configured proxy (*e.g.* it is in an internal network), you can deactivate it by temporarily setting the HTTP_PROXY variable to “”:

```
$ cd ~/auditorium-scripts
$ HTTP_PROXY="" ./list_projects.py --controller 192.168.1.3 --login openbach
```